UniToolBench: A Benchmark for Tool-Augmented LLMs in Cross-Domain, Universal Task Automation

Xiaojie Guo[1], Yang Zhang[1], Bing Zhang[1], Ryo Kawahara[2], Mikio Takeuchi[2], Yada Zhu[1],

[1]IBM Research, [2]Tokyo Research Lab, IBM Research,
`{xiaojie.guo,yang.zhang2,bing.zhang}@ibm.com`
`{yzhu}@us.ibm.com`, `{ryokawa,mtake}@jp.ibm.com`

Abstract

Recent advancements in Large Language Models (LLMs) have enabled autonomous agents to decompose complex tasks, select appropriate tools, and execute structured workflows. However, a key challenge in this field is the lack of a universal, large-scale, and cross-domain benchmark to systematically evaluate LLMs' ability to reason over and utilize interconnected tools for automation. Existing benchmarks, such as TaskBench, focus on manually curated tool graphs for benchmark generation, which lack scalability and domain diversity. To address this, we propose UniToolBench, a benchmark that incorporates automated tool graph construction by formulating link prediction as a probabilistic task, instead of relying on categorical LLM outputs. Furthermore, we introduce a confidence-based beam search sampling strategy to select highconfidence tool dependencies, ensuring more structured and semantically coherent subgraphs for evaluation. Through extensive experiments on multiple datasets, we demonstrate that while LLMs show promise in tool selection, significant challenges remain in parameter prediction and handling complex tool dependencies.

1 Introduction

Recent advancements in Large Language Models (LLMs) (Shen et al., 2024; Hong et al.; Wen et al., 2023; Park et al., 2023) have significantly expanded their ability to reason, plan, and interact with external tools, making them a promising foundation for autonomous task automation. LLM-empowered autonomous agents have emerged as a novel paradigm, capable of decomposing complex user instructions into structured actions and leveraging various tools to accomplish tasks. In many real-world scenarios, such as API orchestration, data processing pipelines, or robotic control, these agents must operate within a structured environment where multiple tools are interconnected (Yuan et al., 2024). Understanding the relationships between these tools and utilizing them efficiently is crucial for effective task execution. A key challenge in advancing this field is the lack of a systematic and standardized benchmark to assess the ability of LLMs in task automation comprehensively. As a result, developing such a benchmark has become essential to drive progress and support further research in this area.

To address the above challenge, TaskBench (Shen et al., 2023) is introduced as a framework to evaluate the capability of LLMs in task execution. The task execution process is divided into three critical stages: task decomposition, tool selection, and parameter prediction. To generate a benchmark like this, a fundamental requirement for such systems is the construction of a *tool graph*, which encodes dependencies and interactions among tools. Based on the *tool graph*, a back-instruct method can be used to generate high-quality user requests and task decomposition steps. However, real-world scenarios brought many more challenges towards complicated task execution tasks: (1) **Greater diversity and quantity in tools**: Real-world automation tasks often require tools from multiple domains to be combined effectively for a complex task, making it essential to handle a significantly larger tool set. Unlike constrained benchmarks with a limited number of predefined tools, real-world applications involve tools with varying functionalities, input-output formats, and execution constraints. (2) **More complexity in tool dependencies**: Beyond simply selecting the correct tools, models must understand the intricate dependencies among them. In real-world scenarios, tools often exhibit hierarchical, conditional, and multistep dependencies, where the output of one tool dynamically affects the parameters or execution of another. Thus, a universal, cross-domain evaluation benchmark that includes more and diverse tools and tool dependencies in this area has become an urgent need.

Existing approaches, such as TaskBench (Shen et al., 2023), manually design tool graphs based on domain expertise, ensuring a well-structured and meaningful representation. However, as the number of tools available increases, the manual construction and maintenance of tool graphs becomes impractical. The complexity of inter-tool relationships, evolving functionalities, and the vast search space further exacerbate this challenge, making it essential to develop an automated approach to tool graph construction. To address these challenges, we propose a novel **LLM-driven approach for automatic tool graph construction**. Our method formulates the problem as a link prediction task, where an LLM determines whether a directed edge should exist between two tools based on their functionalities and dependencies. Instead of relying on direct categorical output, we extract the final-layer logits corresponding to the tokens `"yes"` and `"no"`, normalize them into probability values, and determine the existence of the link through an optimal thresholding mechanism. This probabilistic formulation enhances robustness and adaptability in tool graph generation. To account for the heterogeneity of inter-tool relationships, our approach explicitly prompts the LLM to consider different aspects of tool connections, thus making it generalizable across different domains.

Beyond graph construction, an additional challenge arises in *sampling subgraphs* for generating training data. Random sampling sub-graphs from an LLM-based tool graph often leads to suboptimal tool sequences, as it does not account for the likelihood of tool dependencies occurring naturally. This can introduce a high level of noise, where many sampled subgraphs contain irrelevant, weak, or even conflicting tool interactions, reducing the overall quality of the dataset. To this end, we introduce a **confidence-based beam search** strategy. This method selects high-confidence edges within the tool graph to form a structured subgraph, ensuring a more reliable and semantically meaningful evaluation benchmark. The proposed confidence-based beam search prioritizes high-probability tool connections, effectively filtering out spurious or implausible edges.

1.1 Contributions

Our work makes the following key contributions:

- **Automated Tool Graph Construction**: We propose an LLM-driven approach for link prediction in tool graphs, leveraging logit-based probability normalization and threshold optimization.

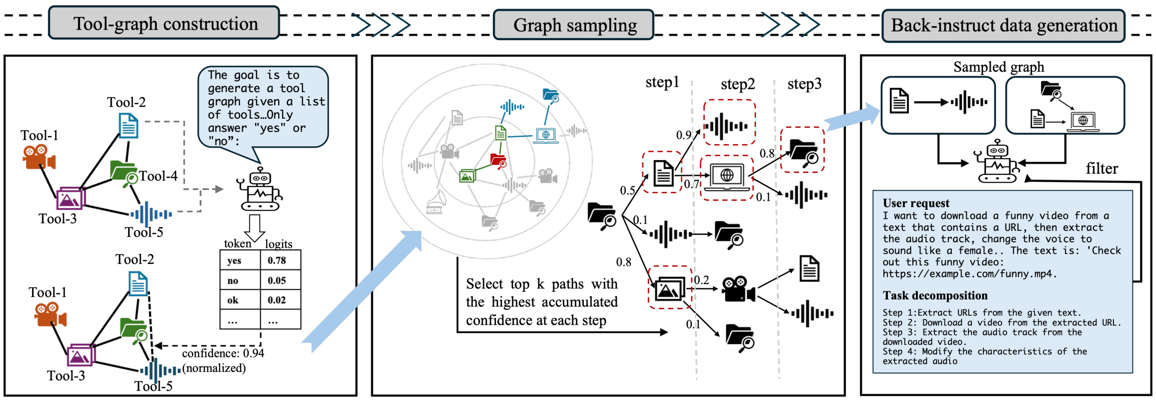
- **Confidence-Based Beam Search for Graph Sampling**: We introduce a structured sampling method that prioritizes high-confidence links, ensuring that selected subgraphs maintain strong semantic coherence.

- **Comprehensive experiments**: We perform extensive evaluations on multiple datasets, including Huggingface, DailylifeAPI, Multimedia, and Universal. Our experiments benchmark four popular models for tool selection, parameter prediction, and graph construction accuracy.

2 Related Work

The integration of external tools with LLMs has been explored through various approaches, focusing primarily on structured execution, dynamic tool selection, and reasoning-based orchestration. Early methods used heuristic or rule-based systems to select tools based on task requirements. More recent works, such as ReAct (Yao et al., 2022) and Toolformer (Schick and Schütze, 2023), enable LLMs to select and invoke tools based on contextual needs dynamically.

Various benchmarks have been designed to evaluate how LLMs interact with external tools, with a primary focus on tool invocation and API utilization. API-Bank (Li et al., 2023) and ToolBench (Qin et al., 2023) generate tasks from API documentation, but their template-driven sampling approach can lead to limited logical coherence in task construction. MetaTool (Huang et al., 2023) and ToolAlpaca (Tang et al., 2023) take a different approach by employing template-based tool generation to determine whether a tool is necessary for a given task. Yet, it lacks the ability to model complex dependencies between tools. AgentBench (Liu et al., 2023), on the other hand, evaluates LLMs within simulated environments, emphasizing agent-like decision-making rather than structured tool usage. Additionally, ToolQA (Zhuang et al., 2024)


Tool-graph construction
Graph sampling
Back-instruct data generation
Tool-1
Tool-2
Tool-3
Tool-4
Tool-5
The goal is to generate a tool graph given a list of tools…Only answer "yes" or "no":
token
logits
yes
0.78
no
0.05
ok
0.02
…
…
confidence: 0.94 (normalized)
step1
step2
step3
0.9
0.8
0.7
0.5
0.1
0.8
0.2
0.1
0.1
Select top k paths with the highest accumulated confidence at each step
Sampled graph
filter

User request
I want to download a funny video from a text that contains a URL, then extract the audio track, change the voice to sound like a female.. The text is: 'Check out this funny video: https://example.com/funny.mp4.

Task decomposition
Step 1:Extract URLs from the given text.
Step 2: Download a video from the extracted URL.
Step 3: Extract the audio track from the downloaded video.
Step 4: Modify the characteristics of the extracted audio

Figure 1: The pipeline of generating UniToolBench consists of three key steps: (1) Automatic Tool Graph Construction, (2) Sub-Tool-Graph Sampling, and (3) Back-Instruct Data Generation.

and GPT4Tools (Yang et al., 2024) focus on assessing task completion accuracy and question-answering performance, but their scope is often restricted to specific domains.

TaskBench (Shen et al., 2023) introduced a structured evaluation benchmark for assessing tool-use capabilities in LLMs, but it requires manually curated tool graphs, which become infeasible at scale. Our work addresses this limitation by introducing an automated tool graph construction framework based on LLM-driven link prediction, reducing reliance on human annotations while maintaining structured tool utilization.

3 Methodology

In this section, we introduce our pipeline for data generation, which consists of four main components: LLM-assisted tool graph construction, confidence-based sub-tool-graph sampling, and back-instruct data generation.

3.1 Overall Pipeline of Data Generation

Our methodology follows a structured pipeline to generate high-quality datasets for evaluating tool-using capabilities in LLMs. The pipeline consists of three key steps as shown in Figure 1. In the first step, given a set of tools, along with their descriptions, input-output format, and parameter requirements, as shown in Figure 2, the goal is to construct a tool graph to include all the possible dependencies among tools. After obtaining the overall tool graph, we perform confidence-based beam search sampling in the second step to extract high-confidence subgraphs. This step ensures that selected subgraphs contain highly relevant and semantically meaningful tool interactions. In the third step, given each sampled sub-tool graph, we use back-instruct to generate structured user requests, task steps, and corresponding tool invocation graphs with the help of an LLM, similar to taskBench(Shen et al., 2023). This step is guided by in-context learning techniques and a filtering mechanism to enhance the quality of generated samples.

```
Tool name:
Software_management

Tool description:
Manage a specific software by a specific
instruction

Parameters:
    name: software
    type: string
    desc: The software to manage

    name: instruction
    type: string
    desc: The instruction to manage the
    software by, eg. install, uninstall,
    update etc.
```

Figure 2: The descriptions, input-output format, and parameter requirements of each tool as the input for the graph construction.

3.2 LLM-Assisted Tool Graph Construction

The goal of tool graph construction is to predict the connectivity between tools based on their functionalities. We formulate this as a link prediction problem: Let $\mathcal{T} = \{T_1, T_2, ..., T_N\}$ represent a set of N tools, where each tool T_i has specific tool descriptions, input and output parameter requirements as shown in Figure 2. The tool graph is represented as a directed graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, where $\mathcal{E} \subseteq \mathcal{T} \times \mathcal{T}$ is the set of directed edges indicating valid tool dependencies. Given a pair of tools $(T_i, T_j) \in \mathcal{T} \times \mathcal{T}$, the goal of link prediction is to determine whether a directed edge $e_{ij} \in \mathcal{E}$ should exist. In addition, for each edge e_{ij}, a confidence score c_{ij} will also be derived.

To model this prediction task, we utilize a large language model (LLM) that, given a tool pair (T_i, T_j), estimates a probability score $P(e_{ij})$ representing the likelihood of an edge existing between T_i and T_j. Instead of directly relying on categorical outputs, we extract the logits ℓ_{yes} and ℓ_{no} corresponding to the tokens "yes" and "no" and compute the probability using a softmax function. Specifically, as shown in Figure. 1 (Step 1), to predict whether there is a link between *Tool-2* and *Tool-5*, we utilize a standardized prompt, which include the information of both tools and ask a LLM whether a directed edge should exist between *Tool-2* and *Tool-5*. The prompt is provided in Appendix A. To achieve more reliable predictions, instead of directly relying on the LLM to generate a categorical "yes" or "no" response, we extract the logits of the `"yes"` and `"no"` tokens from the LLM's last layer and normalize them into a probability value. Because the LLM is not trained on graph structure or link supervision, its scores are not calibrated by design. To improve score reliability, we apply temperature scaling to the raw logits before computing softmax probabilities:

$$e_{ij} = \begin{cases} 1, & \frac{e^{W_o^\top h_T[\text{yes}]/T}}{e^{W_o^\top h_T[\text{yes}]/T} + e^{W_o^\top h_T[\text{no}]/T}} \geq \tau \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where T_i and T_j represent two tools under evaluation for a potential link. The binary variable e_{ij} indicates whether a link exists between T_i and T_j. The final-layer hidden state of the Transformer model is represented as h_T, and W_o is the output projection matrix that maps hidden states to token logits. A predefined threshold τ is used to decide whether a link exists between two tools. This allows us to dynamically construct a tool graph based on LLM assessments while minimizing uncertainty in the generated structure.

3.3 Beam-Search Sampling with Confidence Score

After constructing the tool graph, we perform a confidence-based beam search algorithm to extract high-quality and reliable subgraphs. The confidence of each edge in the tool graph reflects the reliability of the edge, which is derived from the probability of its existence. The goal is to find a set of subgraphs that maximizes overall confidence.

Algorithm 1: Beam-Search Sampling for High-Confidence Subgraph Extraction

```
Input: Tool Graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$ with confidence
       scores $c_{ij}$ for each edge $e_{ij}$. Beam width $B$,
       maximum search depth $D$.
Output: Set of high-confidence subgraphs $S$.
1  Initialize beam set $S = \emptyset$;
2  for each starting node $v \in V$ do
3      Initialize priority queue $Q$ with $(v, 0, \emptyset)$ ;
         // Node, Score, Path
4      while $Q$ is not empty and depth $\leq D$ do
5          Pop $(node, score, path)$ from $Q$;
6          for each neighbor $u$ of node in $G$ do
7              Compute new score
                 $S' = score + P(node, u)$;
8              Append $(u, S', path \cup \{(node, u)\})$
                 to $Q$;
9          Keep top $B$ paths in $Q$ sorted by $S'$;
10     Add best sampled subgraph to $S$;
11 return $S$;
```

The **Confidence-based beam-search sampling algorithm** is designed to extract high-confidence subgraphs from a tool graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, where each edge e_{ij} will have a confidence score c_{ij}. As shown in Algorithm 1, the algorithm starts from multiple candidate nodes and iteratively expands paths while maintaining only the top B high-confidence paths at each depth. A *priority queue* is used to track paths, where the cumulative confidence score along each path determines its priority. At each step, the algorithm explores neighboring nodes, updates the cumulative path confidence, and retains only the best candidates. This ensures that the final selected subgraphs maximize the **sum of link confidence scores**, leading to more reliable tool interactions.

Utilizing a beam search strategy instead of random or exhaustive sampling has several advantages. First, it balances *efficiency and optimality*, as it does not require evaluating all possible paths while still retaining the most promising ones. Second, it ensures *robustness in uncertain environments*, as low-confidence links are naturally filtered out. Finally, it enhances *scalability* for large tool graphs, as it systematically prunes weaker paths and focuses on **high-confidence tool dependencies**, resulting in improved downstream data generation and automation accuracy.

3.4 Back-Instruct Data Generation

Given the sampled high-confidence subgraphs, we generate user requests, task decomposed steps, and tool invocation graphs using an LLM. This process ensures that the generated dataset is diverse and aligned with real-world tool interaction patterns. To improve the quality of generated data, we employ in-context learning techniques by providing example interactions to the LLM.

By iteratively refining the input prompts and providing high-quality examples, we enhance the consistency and correctness of the generated data. The prompt for back-instruct generation is provided in Appendix B. The back-instruct step ultimately ensures that the dataset effectively represents real-world tool automation scenarios. To ensure high-quality data, we utilize LLMs as a filter to check the alignments between the generated data and the sampled tool subgraph and filter those with poor alignments.

4 Experiment

We generate the proposed benchmark dataset with LLaMA-3.2-70B and evaluate it along several dimensions: confidence score calibration, tool-graph construction accuracy, sampled subgraph and data quality, and model performance on the benchmark.

4.1 Sources of tools

We explore four sources of tools, denoted as Huggingface, DailylifeAPI, Multimedia, and UniTool. The first three datasets are single-domain and collected from (Shen et al., 2023), with a ground-truth tool graph for validating the effectiveness of the proposed automatic graph construction method. In addition, we collect Unitool, a cross-domain extensive tool collection that includes the first three tool sources, as well as selected tools from ToolMM-Bench (Wang et al., 2024). It is used to form a universal tool graph for generating the UniTool-Bench dataset.

- **Huggingface:** Hugging Face offers an extensive collection of AI models that support a wide range of tasks across different modalities, including text, vision, audio, and video. It consists of 23 tools and 225 human-verified connections for the automatic construction and validation of tool graphs.

- **DailylifeAPI:** The type of dependencies among tools here is predominantly temporal, where two APIs have a successive order if they are connected. It consists of 40 tools and 1,560 human-verified connections.

- **Multimedia:** Multimedia tools provide more user-centric features, such as file downloaders, video editors, and others. The policy for tool connections is the same as the Hugging Face domain. It includes 40 tools and 449 edges.

- **UniTool** To construct a universal cross-domain large tool graph, we collected tools from the three datasets mentioned above, as well as selected tools from ToolMM-Bench (Wang et al., 2024). It includes 113 task categories, each corresponding to a specific example tool or API. Since these tools/APIs are also derived from HuggingFace, we carefully removed overlaps with the previously mentioned HuggingFace tools, resulting in 197 tools.

4.2 Calibration Analysis on Confidence Score

To evaluate the reliability of confidence scores produced by LLMs for link prediction, we conduct a calibration analysis. This analysis assesses whether the predicted scores can be interpreted as meaningful probabilities of link existence. To measure calibration quality, we use the Expected Calibration Error (ECE). Predicted node pairs are grouped into bins based on their confidence scores (i.e., into 10 or 20 quantile bins). For each bin, we compare: (1) the average predicted probability of the bin; and (2) the empirical accuracy, i.e., the proportion of actual links in that bin. ECE is computed as the weighted average of the absolute difference between these two quantities across all bins.

Figure 3 shows calibration curves across the Huggingface, Multimedia, and Dailylifeapis datasets. Among the datasets, the DailylifeAPIs toolset achieves the best calibration with a low Expected Calibration Error (ECE) of 0.020, suggesting high-confidence estimates that align well with actual link reliability. The Huggingface and Multimedia toolset also shows good calibration (ECE = 0.062 and 0.112), with the calibrated curve tracking the diagonal across most probability bins, though with slightly higher variance at low-confidence predictions.

4.3 Automatic Tool Graph Construction and Evaluation

To validate the effectiveness of the LLM-based automatic graph construction, we compare the constructed graph with a human-maintained graph. Here, we utilize Huggingface, DailylifeAPI, and Multimedia to create three individual tool graphs, validating the effectiveness of the proposed method.

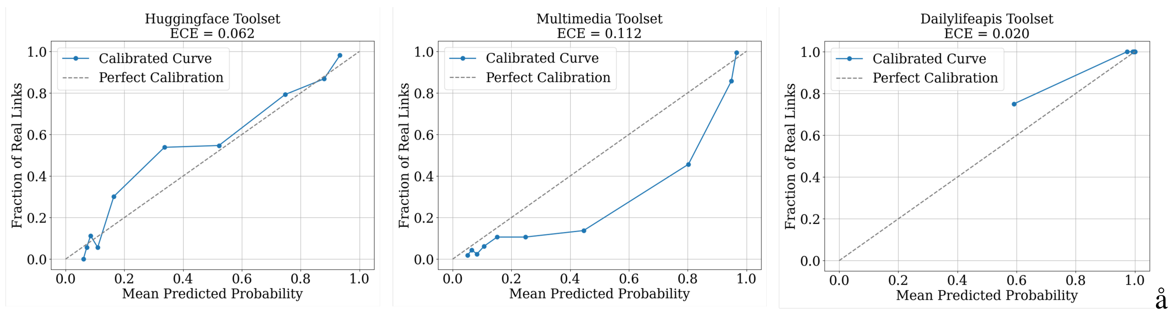


Figure 3: Temperature-Scaled Calibration Curve in three datasets

The graph construction can be regarded as a link prediction task; thus, **precision** and **recall** are used:

- **Precision:** The fraction of correctly predicted edges among all predicted edges. The lower precision results in a higher number of false positives, which means the constructed graph contains many unreasonable links.

- **Recall:** The fraction of correctly predicted edges among all true edges. The lower recall means a large false negatives, which means the constructed graph lost many existing links.

Table 1: Precision and Recall for the graph construction from each tool source.

Dataset	Precision↑	Recall↑	Time(min)
Huggingface	0.81	0.75	63
DailylifeAPI	0.99	0.98	216
Multimedia	0.85	0.77	210

The evaluation results in Table 1 demonstrate the effectiveness of the automatic LLM-based graph construction by comparing it with manually constructed tool graphs. Specifically, DailylifeAPI achieves the highest performance with a precision of 0.97 and a recall of 0.98 at a low threshold of 0.02. This suggests that the LLM confidently predicts the tool connections with minimal false positives and false negatives, likely due to the strong structural consistency in the dataset. In contrast, the tool graph construction of Huggingface and Multimedia shows lower precision (0.81 and 0.85, respectively) and recall (0.75 and 0.77), indicating more challenging tool dependencies, where a type match is required between two connected tools.

In addition, we list the time cost of the graph construction process for each toolset. For a toolset with 40 tools, e.g., DailylifeAPI and Multimedia, the total time cost is around *210* minutes, and for a toolset with 20 tools, e.g., Huggingface, the time cost is *63* minutes, which vastly improves the efficiency of tool graph construction and provides the possibility for much larger graph construction.

From the UniTool, we can generate a large tool graph with 13,137 edges and 197 nodes, reflecting multiple complex tool dependencies. From this graph, subgraphs will be sampled as seeds for benchmark data generation.

4.4 Sampled sub-graphs and Evaluation

The quality of sampled subgraphs is crucial for ensuring the diversity and representativeness of the generated dataset. To evaluate the effectiveness of the sampling method, we proposed the Node Type Coverage (NTC) and Edge Relation Coverage (ERC) across different constructed graphs.

Node Type Coverage (NTC): This metric measures the fraction of distinct node types present in the sampled subgraph relative to the total number of node types in the original large graph. A higher NTC indicates that the sample encompasses a broader range of node types.

Edge Relation Coverage (ERC): This metric evaluates the fraction of distinct edge relation types present in the sampled subgraph compared to the total number of edge relation types in the large graph. A higher ERC indicates the sample captures more diverse edge relations.

Table 2: Evaluation metrics for subgraph sampling across four datasets.

Dataset	NTC↑	ERC↑
Huggingface	100%	85.19%
DailylifeAPI	100%	54.92%
Multimedia	100%	69.01%
UniTool	92.19%	48.26%

As presented in Table 2, NTC values show that the sampled subgraphs from Huggingface, DailylifeAPI, and Multimedia datasets achieve 100% node type coverage, meaning all node types present in the original tool graph are included in the sampled subgraphs. This suggests that our sampling strategy successfully retains the full range of tool categories across these datasets. However, for UniTool, the NTC drops to 92.19%, indicating that some node types are missing.

The ERC values, which measure the diversity of edge relations in the sampled subgraphs, exhibit more variability across datasets. Huggingface achieves the highest ERC (85.19%), followed by Multimedia (69.01%) and DailylifeAPI (54.92%), while UniTool has the lowest ERC (43.48%), which suggests that while a broad range of node types is covered, some tool links are not fully captured. This may be due to highly specialized or infrequent tool relations that do not appear frequently enough in the sampled subgraphs.

4.5 Back-instruct data synthesis and Evaluation

To illustrate the quality of the generated datasets, we conducted comprehensive human evaluations and LLM-based evaluations. Here we utilize four metrics (Shen et al., 2023): three are used (i.e., Naturalness, Complexity, and Clarity) to assess the quality of the instructions, and one is (i.e., Alignment) to evaluate the tool invocation graph. Each metric is scored on a scale from 1 to 5. The description of each metric as well as data examples can be found in Appendix C.

Table 3 provides insights into how well the generated instructions and tool invocation graphs align with real-world task execution. Overall, the **LLM-based evaluation** produces consistently high scores across all datasets, with Naturalness, Alignment, and Clarity exceeding 4.0 in most cases. This suggests that the LLM-generated instructions generally align well with expected task workflows and maintain coherent tool dependencies. The high Alignment scores, particularly for *DailylifeAPI* (4.974) and *Huggingface* (4.912), indicate that the tool invocation graphs closely match the intended instructions, ensuring practical tool usage.

In addition, three human experts evaluated subsets of the data independently. Agreement was substantial ($\kappa > 0.75$) for clarity and alignment. The average scores across annotators are reported in Table 3. The **human-expert evaluation** scores are aligned well with LLM-based evaluation, except for a general lower score across Complexity. The highly matched evaluation regarding Clarity between LLMs and human experts indicates that while LLM-generated instructions may not always be perceived as entirely natural or complex, they are still straightforward to understand. This is particularly evident in *Multimedia*, where the human-evaluated Clarity score (4.67) closely aligns with the LLM-based score (4.59), suggesting that the generated instructions are well-structured and free from ambiguity. The big gap in Complexity reflects that humans may perceive specific tool sequences as oversimplified or lacking true task complexity.

4.6 Evaluating Models on generated Benchmark

To assess the performance of LLMs on this benchmark, we evaluate tool selection and argument prediction across the four datasets. We employ a similar prompting strategy as in TaskBench (Shen et al., 2023), which guides each model through a structured sequence: first, decomposing user requests into sub-tasks, and then selecting appropriate tools with parameters. For our evaluations, we focus on two categories of models: (1) Open-domain LLMs: DeepSeek-V3 (Liu et al., 2024a), Qwen-72B (Bai et al., 2023), and GPT-4.1 (OpenAI, 2025); (2) Tool-augmented LLMs: ToolACE-2-8B (Liu et al., 2024b) and Hammer2.1-7b(Lin et al., 2024). The results in Table 4 provide insights into how different models perform in terms of tool selection accuracy (**R1, R2, RL**), execution alignment (**Node-F1**), structure preservation (**NED**), and argument correctness (**Arg-F1**).

In domains with more constrained tool usage and less ambiguous instruction formats, such as *DailylifeAPI*, all models perform comparably well. Notably, DeepSeek-V3 achieves the highest Node-F1 (99.73) and Arg-F1 (99.80), even slightly surpassing GPT-4.1 in these metrics. This suggests that for well-scoped, API-centric tasks, smaller models can closely approximate the gold-standard performance.

As task complexity and cross-domain diversity increase, model performance diverges more noticeably. The *UniToolBench* dataset, designed to assess universal and multi-domain tool use, poses significant challenges across all metrics. While GPT-4.1 maintains relatively strong performance (e.g., R1: 66.78, Arg-F1: 72.89), other models such as ToolACE-2-8B and Hammer2.1-7b experience sharp declines in both tool selection and parameter correctness. In particular, ToolACE drops to

Table 3: Evaluation of Benchmark data (Na is for Naturalness, Com is for Complexity, Align is for Alignment)

Dataset	LLM-Based Metrics				Human-Expert Metrics			
	Na↑	Com↑	Alig↑	Clarity↑	Na	Com↑	Alig↑	Clarity↑
Huggingface	4.40	3.71	4.77	4.39	4.31	2.08	4.72	4.72
DailylifeAPI	3.99	3.75	4.97	4.59	4.39	2.71	4.55	4.67
Multimedia	4.29	3.88	4.91	4.91	4.57	2.27	4.72	4.78
UniToolBench	4.07	3.97	4.66	4.48	4.45	2.12	4.70	4.75

Table 4: Evaluation Results: Tool Selection and Parameter Prediction Across Datasets

Dataset	Model	R1↑	R2↑	RL↑	Node-F1↑	NED↓	Arg-F1↑
Huggingface	ToolACE-2-8B	42.16	27.97	37.47	85.40	0.06	41.11
	Hammer2.1-7b	46.43	30.67	39.75	77.65	0.15	38.41
	DeepSeek-V3	47.07	32.00	40.93	86.92	0.04	35.84
	Qwen-72B	46.91	31.23	40.66	84.49	0.07	36.10
	GPT-4.1	56.71	34.92	47.15	87.01	0.05	34.11
DailylifeAPI	ToolACE-2-8B	86.85	79.72	86.44	99.61	0.02	99.64
	Hammer2.1-7b	86.77	79.67	89.95	98.77	0.02	98.11
	DeepSeek-V3	89.57	81.45	88.73	99.73	0.01	99.80
	Qwen-72B	86.99	78.70	86.11	99.33	0.01	99.45
	GPT-4.1	88.95	80.20	87.71	99.56	0.02	99.29
Multimedia	ToolACE-2-8B	41.51	29.25	36.61	96.06	0.04	70.84
	Hammer2.1-7b	43.70	30.34	39.59	94.51	0.08	43.61
	DeepSeek-V3	45.32	32.11	41.88	97.25	0.03	32.75
	Qwen-72B	46.58	32.82	42.41	96.09	0.04	53.48
	GPT-4.1	56.71	35.99	42.67	99.13	0.01	58.63
UniToolBench	ToolACE-2-8B	29.12	12.40	25.09	63.76	0.25	16.26
	Hammer2.1-7b	32.59	15.35	27.86	53.42	0.21	22.37
	DeepSeek-V3	31.54	14.27	26.48	71.42	0.24	21.20
	Qwen-72B	33.84	15.96	28.09	69.18	0.25	25.94
	GPT-4.1	66.78	49.67	62.33	71.99	0.18	72.89

just 29.12 in R1 and 16.26 in Arg-F1, indicating difficulty in managing tool grounding across variable tasks. Despite showing competitive Node-F1 scores in simpler domains, models like Qwen-72B and DeepSeek-V3 exhibit reduced structural and parameter fidelity under the open-ended conditions.

Different models also show varied strengths depending on the evaluation dimension. For example, Qwen-72B maintains relatively stable NED scores across datasets, suggesting consistent structure generation, while DeepSeek-V3 shows higher execution alignment (Node-F1) even when its argument prediction suffers (e.g., Arg-F1 = 32.75 on Multimedia). These discrepancies reveal that excelling in tool chaining does not necessarily imply accurate parameter inference, highlighting the need for fine-grained evaluation beyond end-task correctness. Collectively, these findings affirm the value of UniToolBench as a challenging and diagnostic benchmark for tool-augmented language models in real-world settings.

5 Conclusion

In this paper, we introduced UniToolBench, a universal benchmark for evaluating LLM-driven task automation across diverse domains. Our contributions include an automated tool-graph construction method with logits-based probability calibration and a confidence-guided beam search for subgraph selection. Compared to prior benchmarks, UniToolBench enables richer cross-domain tool interactions, more realistic task scenarios, and stronger tests of generalization. Experiments show that while LLMs perform reasonably in tool selection, they struggle with parameter prediction and complex cross-domain reasoning, with performance dropping sharply on the universal dataset. Although the current toolset is curated and finite, our fully automated pipeline makes UniToolBench naturally extensible, allowing continuous integration of new or community-contributed tools toward open-world settings.

Acknowledgements

We thank MIT-IBM Watson AI Lab's funding support to this work, and valuable feedback and guidance from David Cox, Dan Gutfreund and Kate Souel.

Limitations

While UNITOOLBENCH introduces significant advantages in benchmarking tool-augmented language models, it is important to outline the scope of the current release. Our benchmark covers 197 curated tools across three diverse domains (NLP, multimedia, daily APIs). This curated design is intentional: it ensures reproducibility, fair comparison across models, and scalability beyond prior work such as TaskBench. At the same time, our pipeline is fully automated and naturally extensible, allowing new or community-contributed tools to be incorporated in future versions. Thus, while the present release does not yet capture the full breadth of open-world tool use, it provides a solid and extensible foundation for such exploration.

In addition, our current evaluation abstracts away execution dynamics by assuming that tool APIs are correctly defined and available. This abstraction is consistent with prior benchmarks and allows us to focus on reasoning and tool-graph planning, rather than system-level engineering. Future extensions may incorporate more realistic dynamics such as probabilistic failures, latency-aware evaluation, or human-in-the-loop feedback. These directions will further enrich realism, but do not affect the validity of our current results, which remain reproducible, comparable, and significantly more scalable than existing benchmarks.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework, 2023. *URL https://arxiv.org/abs/2308.00352*.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Zhenqiang Gong, et al. 2023. Metatool benchmark for large language models: Deciding whether to use tools and which to use. *arXiv preprint arXiv:2310.03128*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*.

Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. 2024. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint arXiv:2410.04587*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, et al. 2024b. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

OpenAI. 2025. Gpt-4.1. https://openai.com/index/gpt-4-1. Large language model, version GPT-4.1. Accessed: 2025-10-02.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pages 1–22.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Timo Schick and Hinrich Schütze. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36.

Yongliang Shen et al. 2023. Taskbench: Benchmarking large language models for task automation. *arXiv preprint arXiv:2311.18760*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Chenyu Wang, Weixin Luo, Qianyu Chen, Haonan Mai, Jindi Guo, Sixun Dong, XM Xuan, Zhengxin Li, Lin Ma, and Shenghua Gao. 2024. Mllm-tool: A multimodal large language model for tool agent learning. *arXiv preprint arXiv:2401.10727*, 4.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2023. Empowering llm to use smartphone for intelligent task automation. *arXiv preprint arXiv:2308.15272*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2024. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao et al. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *arXiv preprint arXiv:2401.06201*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2024. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36.

A Prompt for Graph Construction

We utilize the following prompt template for the automatic graph construction process. Each sample is generated through a single prompting. We assign “tool information" and "tool name" to specific fields in the prompt, and then populate the relevant fields to complete the data generation in a single prompt.

```
The goal is to generate a tool graph given a list
of tools, where each node refers to a tool, and
every two tools in the graph can have a
connection if they meet either of the two
conditions: (1) the output type of source tool is
the input type of the target tool, or (2) they
are used in a specific order, for example, one
tool can only be utilized after the completion of
another tool.

You have access to the following tools
information. Each tool description contains the
tool name (id), tool descriptions and its
parameters.

<tool information>

Is there a link between the source tool
<source_tool> and the target tool <target_tool>?
Only answer "yes" or "no". The answer is:
```

B Prompt for Back-Instruct

We utilize the following prompt template for the “Back-Instruct" Data generation system. Each sample is generated through a single prompting. We assign “instruction," “tool invocation graph," and “self-critics" to specific fields in the prompt, and then populate the relevant fields to complete the data generation within a single prompt.

C Dataset quality evaluation

To illustrate the quality of the generated datasets, we conducted comprehensive human evaluations and LLM-based evaluations. Here we utilize four metrics (Shen et al., 2023): three are used (i.e., Naturalness, Complexity, and Clarity) to assess the quality of the instructions, and one is (i.e., Alignment) to evaluate the tool invocation graph. Each metric is scored on a scale from 1 to 5.

- **Naturalness**: This metric assesses the plausibility and coherence of the generated instructions, focusing on whether the tool dependencies reflect realistic and commonly used workflows. A higher score indicates that the instructions align with intuitive task structures and real-world application scenarios.
- **Complexity**: This metric measures the intricacy of the instructions by considering factors such as the number of tools involved, the overall task difficulty, and the interdependencies among tools. Higher scores indicate instructions that require multiple tools and exhibit sophisticated interactions among these tools.
- **Alignment**: This measures how well the tool invocation graphs align with the instructions, specifically evaluating whether the graphs effectively fulfill the user’s commands.
- **Clarity**: This metric assesses how clearly the instructions are expressed. Well-written instructions that are easy to follow and free from ambiguity will receive a higher score. The clarity of the instructions is crucial for ensuring that the intended task can be easily understood and executed.

Table 5 shows two examples of generated user requests and decomposed steps with different numbers of tools in different domains.

Table 5: Examples of generated user requests and corresponding task decompositions from UNITOOLBENCH.

User Request	**Decomposed Task Steps**
I want to order a pizza to be delivered to 123 Main St using Uber Eats, then order a taxi to the same location using Uber, and finally send an SMS to 123-456-7890 to notify that the food has been ordered and a taxi is on the way.	1) Call **order_food_delivery** tool with food: 'pizza', location:'123 Main St', platform: 'Uber Eats'. 2) Call **order taxi** tool with location: '123 Main St', platform: 'Uber'. 3) Call **send_sms** tool with phone number: '123-456-7890', content: 'Your food has been ordered and a taxi is on the way.'
I have an audio file example.wav containing a speech, and I want to analyze the sentiment of the speech.	1) Call **audio_to_text** tool with audio: 'example.wav' and output: 'transcribed text'. 2) Call **text_sentiment_analysis** tool with text: 'transcribed text', output: 'sentiment result'.

```
Based on the above tool graph, please be skillful
to generate the according task steps, user request
and tool invoking graph. Requirements:

1. The number of task steps must be the same as the
nodes. Each tool node can only be used once; each
task step corresponds to a tool node in the tool
graph and tool invoking graph.
2. The generated user request should be somewhat
clear, self-contained (user-specified text, image,
video, audio, content should be contained in the
request) and practical (help users solve a
practical problem).
3. The task steps must be strictly aligned with the
tool graph (nodes and edges) and reasonable, the
tool invoking graph must align with task steps,
also with the given tool graph.
4. If there are edges, the user request just can be
decomposed into task steps solved by the tool
invoking graph.
5. If need image/audio/video resources in user
request, please use files
'example.[jpg/mp4/wav/png]'.
6. The dependencies among task steps must align
with the edges of tool graph and tool invoking
graph.
7. The number and types of tool parameters in the
generated tool invoking graph need to be consistent
with the pre-defined input/outputs types of the
tools.

Here is an example of generated task steps, user
request and tool invoking graph.

<task steps>
<user request>
<tool invoking graph>

Now please generate your result (with random seed
{seed}) in a compact JSON format:
```